Programming in Python

Dr. Damitha Karunaratna

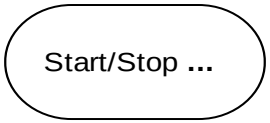
ඇල්ගොරිතමයක් (Algorithm) යනු කුමක් ද?

- ගැටලුවක් විසඳීම සඳහා අනුගමනය කරන ක්‍රමවේදයක් ඇල්ගොරිතමයක් (Algorithm) ලෙස හැඳින්වෙයි.
- ඇල්ගොරිතමයක් රූපමය ආකාරයෙන් (Graphical) හෝ ලිඛිත ආකාරයෙන් (Textual) දැක්විය හැකි ය.
  - රූපමය ආකාරයෙන් - **ගැලීම් සටහන්**
  - ලිඛිත ආකාරයෙන් - Pseudo Code

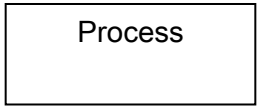
ගැලීම් සටහන් (ANSI Standard)

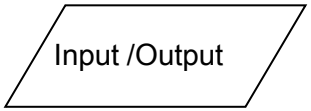
ගැලීම් සටහන් ඇඳීම සඳහා භාවිත කරන ප්‍රධාන සංකේත

Terminal Symbol

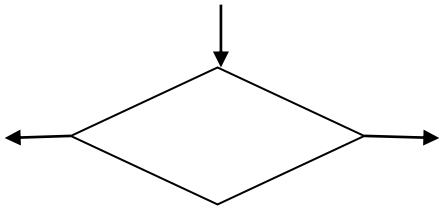


Used to represent arithmetic/ data movement



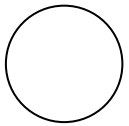


ආරම්භය/ අවසානය



නීරණ කොටුව

දන්න ගලා යාම

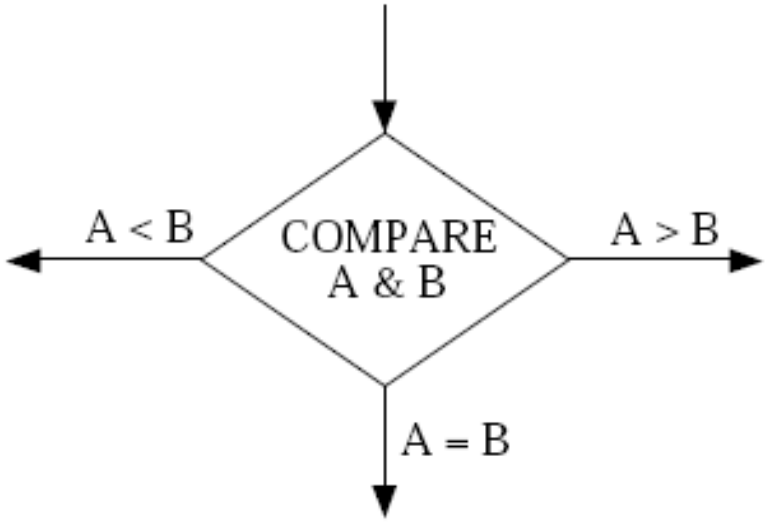


සම්බන්ධක

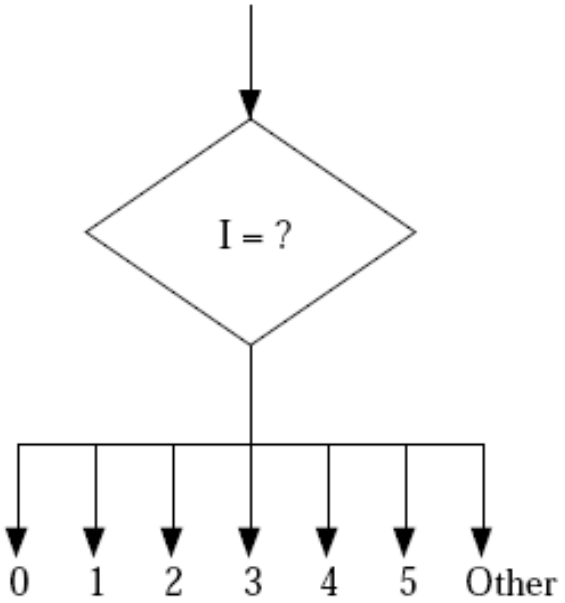
ගැලීම් සටහන් (ANSI Standard) ....

- Processing Box
  - When more than one arithmetic and data movement instructions are to be executed consecutively, they are normally placed in the same processing box and they are assumed to be executed in the order of their appearance.
- The normal flow of flowchart is from top to bottom and left to right.
- Flow lines are usually drawn with an arrowhead at the point of entry to a symbol.

Multi-way branching



(b) Three-way branch

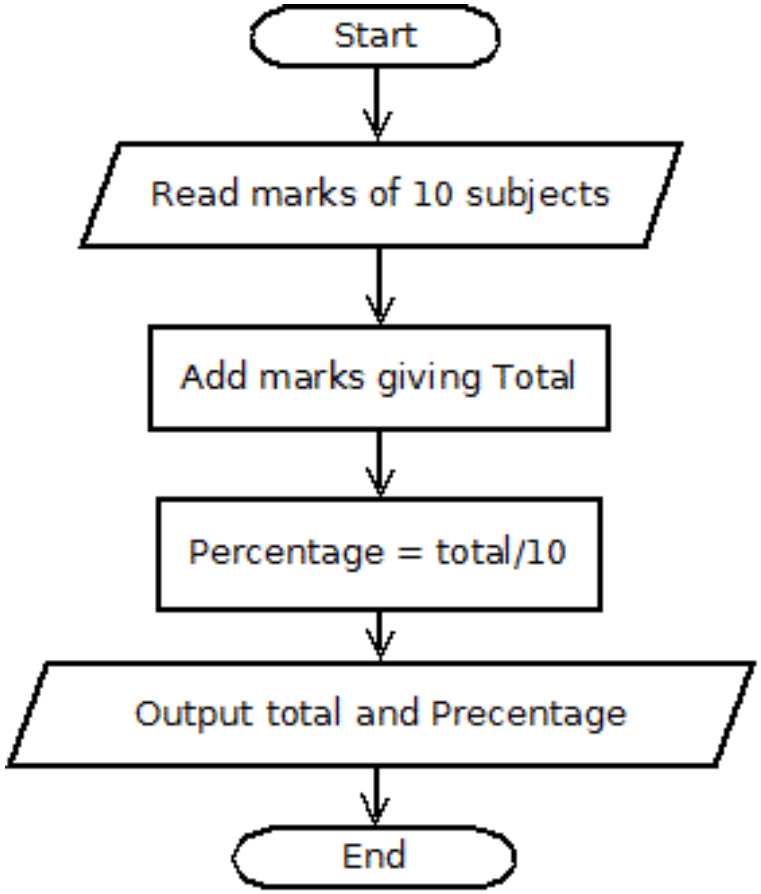


(c) Multiple-way branch

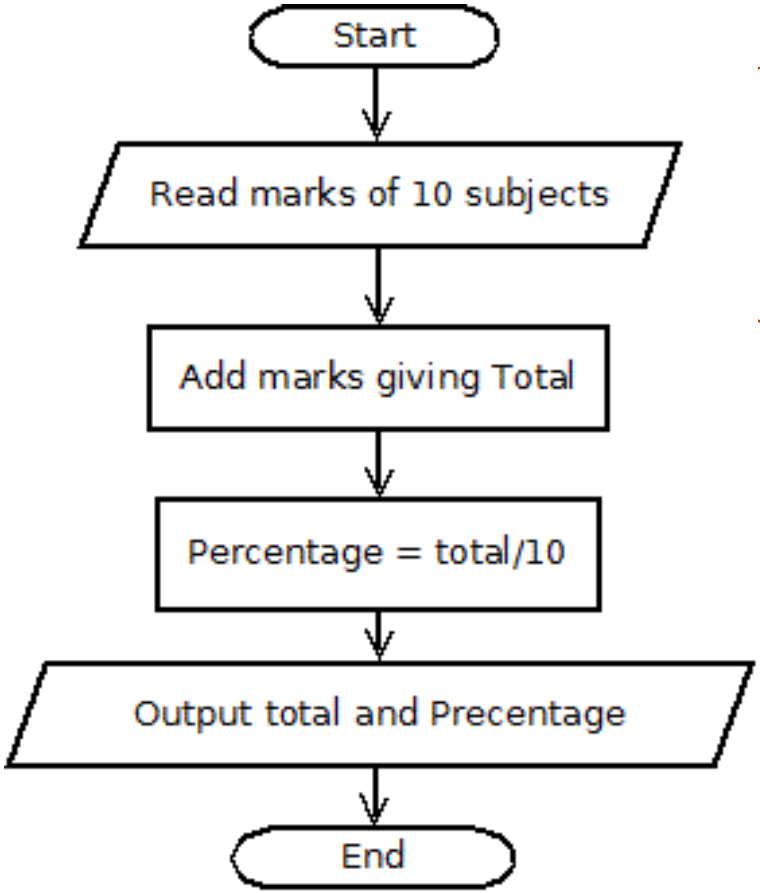
Example

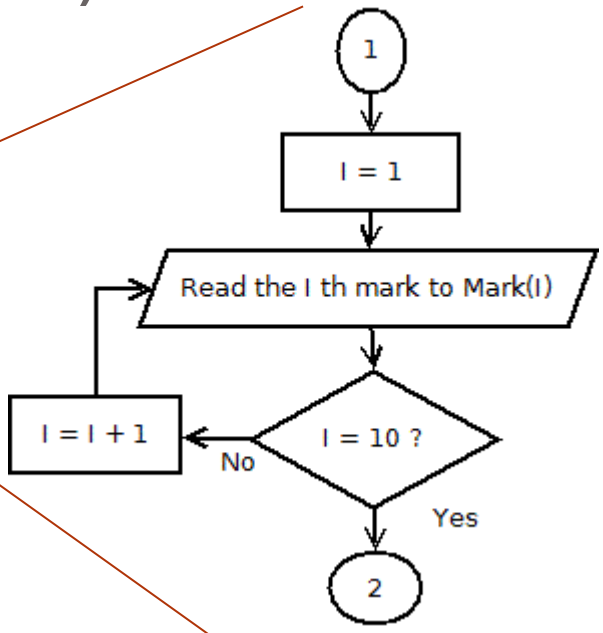
- Draw a flow chart to add marks of ten subjects obtained by a student at an examination and to print the total marks and the average marks obtained by the student.

Example (Macro Level – Solution 1)

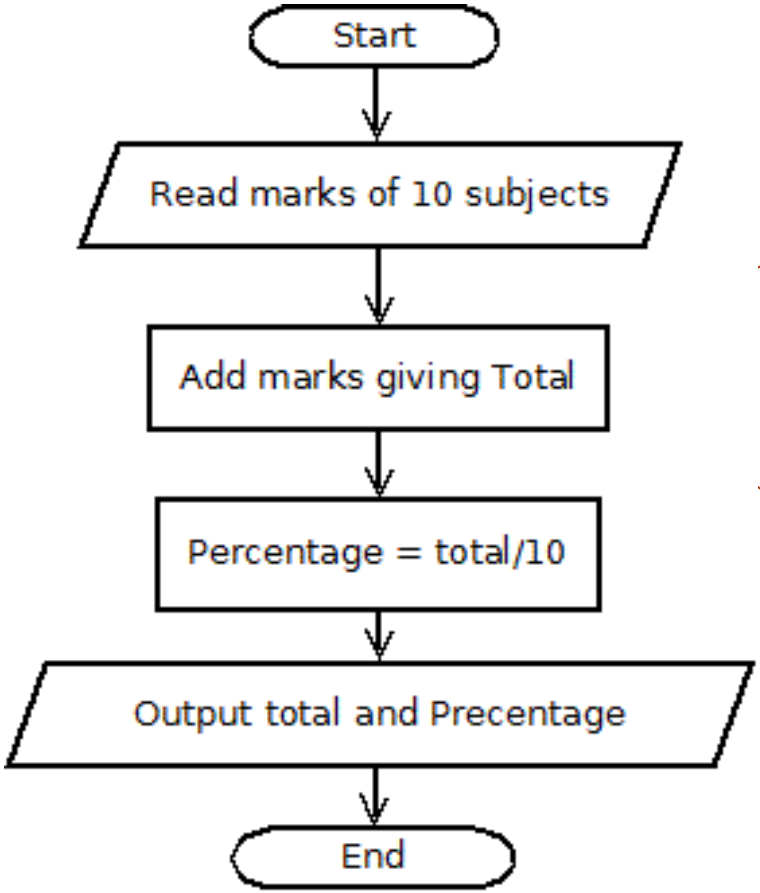


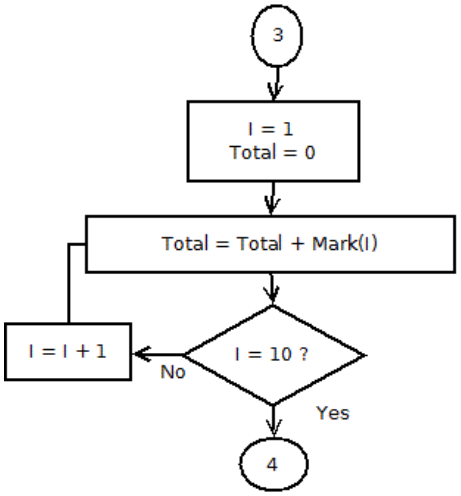
Example (Micro Level)



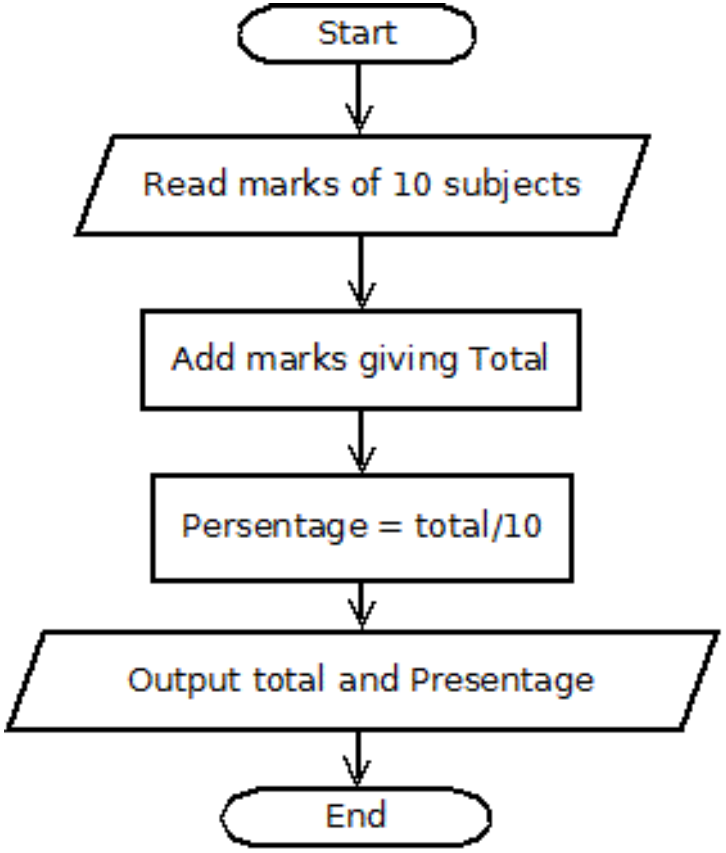


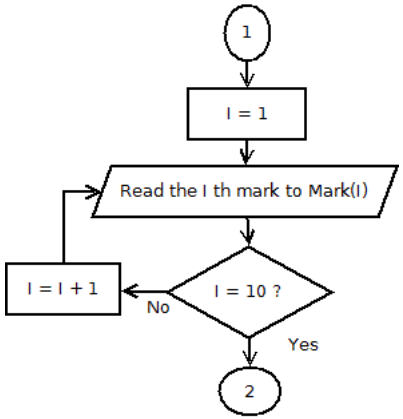
Example (Micro Level)

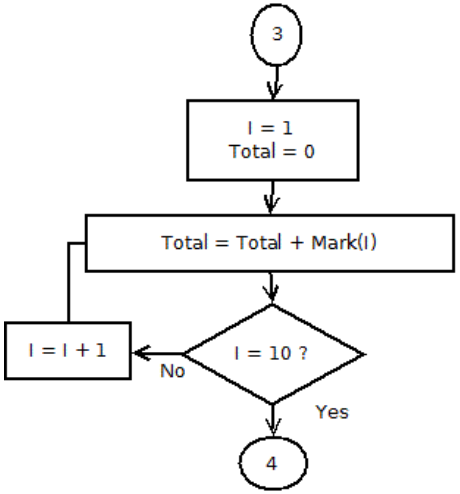




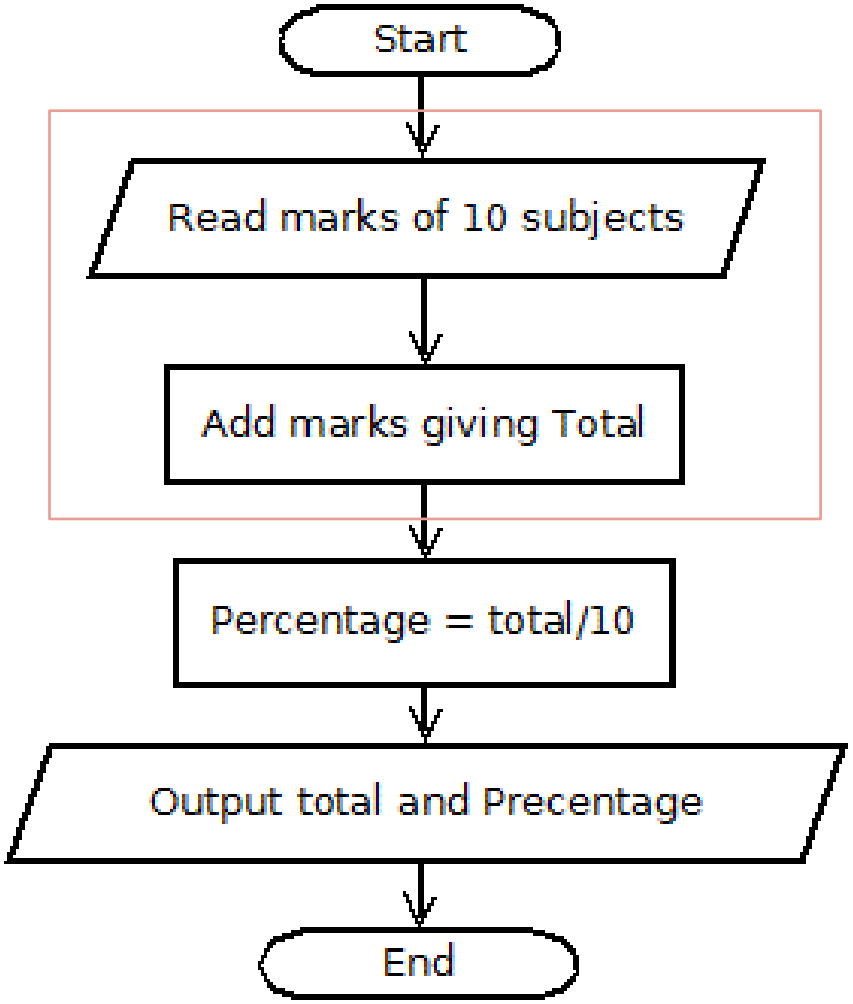
Example (Micro Level)







Example (Macro Level – Solution 2)



ලිඛිත ආකාරයෙන් - Pseudo Code

- **සංඛ්‍යා දෙකක එකතුව ලබා ගැනීම සඳහා කල්පිත ක්‍රමලේඛය (**Pseudo Code**)**

ආරම්භය (Begin)

පළමු සංඛ්‍යාව හා දෙවන සංඛ්‍යාව ඇතුළත් කරන්න

එකතුව = 1 සංඛ්‍යාව + 2 සංඛ්‍යාව

එකතුව ප්‍රතිදානය කරන්න.

අවසානය (End)

පරිගණක භාෂාවන්ගේ පරිනාමය

- පළමු පරම්පරාවේ පරිගණක භාෂා (Machine Languages)
- දෙවන පරම්පරාවේ පරිගණක භාෂා (Assembly Languages)
- තුන්වන පරම්පරාවේ පරිගණක භාෂා (High Level Languages)
- හතරවන පරම්පරාවේ පරිගණක භාෂා (Artificial Languages)

High-level Vs Low-level Programming Languages

- high-level languages:
  - Take less time to write,
  - Typically shorter than an equivalent low level programme.
  - Easier to read
  - Portable - can run on different kinds of computers with few or no modifications.
    - Low-level programs can run on only one kind of computer and have to be rewritten to run on another.

Low-level and High level Programming Languages?

- Every computer is based on a set of instructions built into the hardware(machine code). These instructions, in general, are
  - Simple.
  - specific to the hardware of the particular type of the computer.
  - designed for the hardware but not for humans to follow.
  - directly executable by a computer's central processing unit (CPU).

    A machine language can be considered as a hardware-dependent programming language.

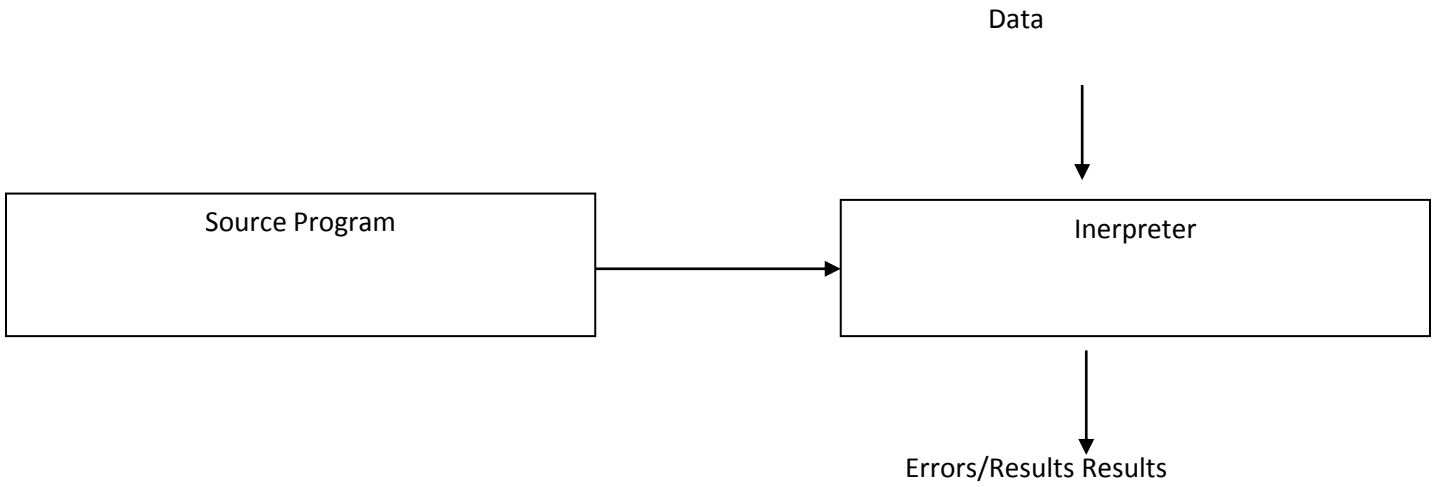
What is a program & a programming language?

- Generally, a program can be considered as a ***sequence of instructions*** associated with a **name** prepared for the computer to carry out a *specific task(s)*.
- A **programming language** is
  - an artificial language designed to communicate instructions to a computer.
  - a notation for writing programs.
  - a notation for specifying computations or algorithms.

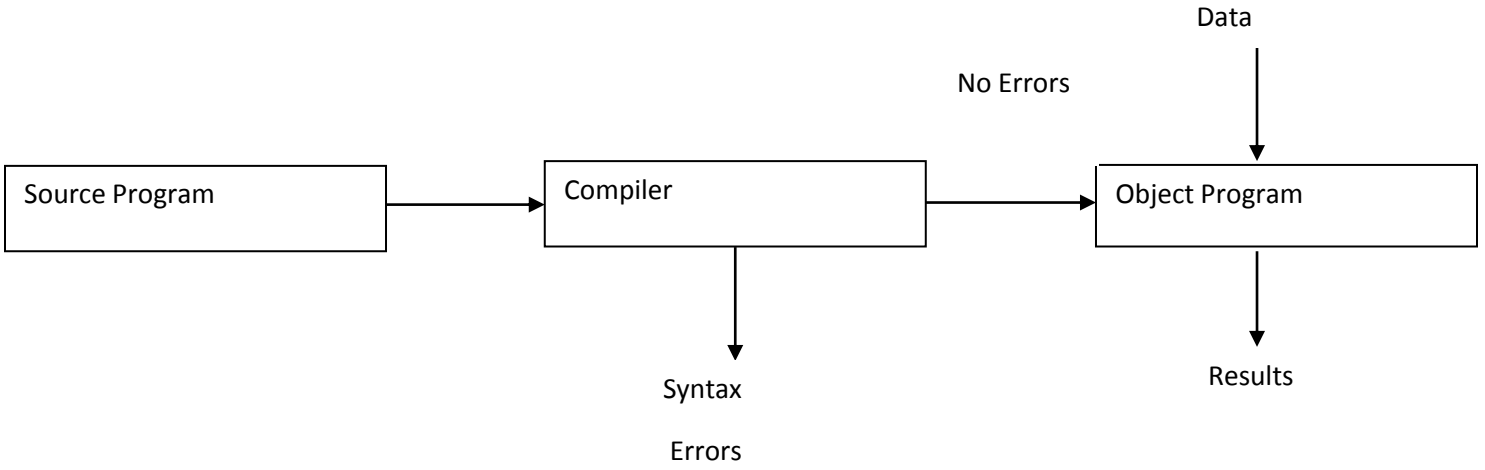
පරිගණක භාෂා පරිවර්තක මෘදුකාංග (Programming Language Translators) ?

- ?????

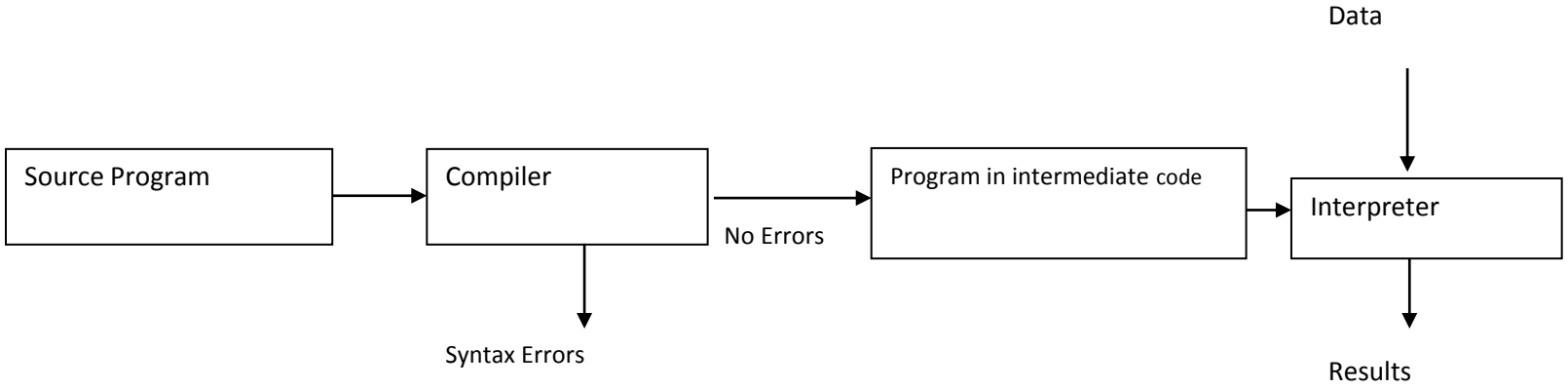
අර්ථ විනාායක (Interpreters)



සම්පාදක (Compilers)



සම්පාදක / අර්ථ විනාශක



Programming

- Few basic instructions appear in every language , but how these instructions are coded vary among languages:
  - **input:** Get data from the keyboard, a file, or some other device.
  - **output:** Display data on the screen or send data to a file or other device.
  - **Computation :** Perform operations on data either by using operations provided by the language or calling functions..
  - **conditional execution:** Check for certain conditions and execute the appropriate code.
  - **repetition:** Perform some action repeatedly, usually with some variation.

Programming …

- Programming can be viewed as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed with one of the instructions provided by the language.

Debugging Programs

- Types of errors that can occur in a program:
  - syntax errors – errors in the structure of a program. Python interpreter can not execute programs with syntax errors.
  - runtime errors(exceptions) – errors that appear when the program is in execution.
  - semantic errors (logical errors) – errors in the logic. Program will run without any errors, but will produce incorrect results.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

Edsger W. Dijkstra

භාෂාවල කාරක රීති (Syntax) සහ ශබ්දාර්ථ (Semantics)

- පරිගණක භාෂා නිර්වචනය කිරීම සඳහා ප්‍රධාන සංඝටක දෙකක් අවශ්‍ය වෙයි. මේවා නම්
  - කාරක රීති **(Syntax)**
    - **Syntax of the tokens :** the basic elements of the language
    - **Syntax of the structure :** the way the tokens are arranged
  - ශබ්දාර්ථ **(Semantics)**

කාරක රීති (Syntax)

- ක්‍රමලේඛ භාෂාවක කාරක නීති මගින් ක්‍රමලේඛ භාෂාවේ ඇති ව්‍යාකරණ නීතිවලට අනුව සංක්ත සම්බන්ධ කළ යුතු ආකාරය නිර්වචනය කරයි.
  - මෙම කාරක රීතිවලින් නිර්වචනය වන්නේ සංකේතවල ව්‍යුහයන්ගේ නිවැරදිතාව පිළිබඳ පමණක් වන අතර ඒවායේ අර්ථය පිළිබඳ සැලකිල්ලක් නොදැක්වීමයි.

උදාහරණ

Cat is a reptile

ශබ්දාර්ථ (Semantics)

මෙමඟින් භාෂාවක ඇති ව්‍යාකරණානුකූල ව්‍යුහයන්ට අනන්‍ය අර්ථ පවරනු ලබයි. එබැවින් භාෂාවකින් සම්පාදනය කළ වැඩසටහනක් ක්‍රියාත්මක කිරීමේදී අනුගමනය කළ යුතු චර්යාව ශබ්දාර්ථ මඟින් නිර්වචනය වෙයි

Python

- Python is open source software.
- Python was initially developed by *Guido van Rossum*.
- It was first released in 1991.
- Python is a high level, general purpose, multiplatform, interpreted language.
- It is designed to he increase programmer productivity and code readability.

Programming in Python

- What you should have?
  - Python installation and
  - a text editor
- Python software can be downloaded from http://www.python.org/download
- Python comes with its own **integrated development environment** (*IDE*), which is quite nice and sufficient for the beginning

Installing python for NetBeans

- Go to Tools | Plugins and open the Settings tab. Add the following site as a new update center.

  http://deadlock.netbeans.org/hudson/job/nbms-and-javadoc/lastStableBuild/artifact/nbbuild/nbms/updates.xml.g

- Go to the Available Plugins tab and select Python plugin, then click on the **install** button.
- Add to the classpath after right-clicking a project and choosing Properties

Practical Session 01

- Objectives :
  - Create, store and run a python program by using the python editor (IDLE).

  - Create a folder to store all your programs (say examples).
  - Open the Python IDLE
  - File -> New Window
  - Add the following text

    ```
    # program - 01
    print("My first program")
    ```
  - Save the program with the name program01.py
  - Execute the program

Using the Python Interpreter

- Python is an interpreted language.
- Two ways to use the interpreter:
  - **Interactive mode :** type Python statements and the interpreter displays the result immediately.
- **Script mode** : store code in a file and use the interpreter to execute the contents of the code in the file.

Practical Session 02

- Objectives :
  - Learn different ways of using the Python Interpreter.
  - Type the following two command at the Python command line

```
# program - 01
print("My first program")
```

  - Store the previous two statements in a file and execute the file in the script mode.

Practical Session 03

- Help on commands

  help(input)

- A program has to written to add two integers and to print the total. The numbers are given from the keyboard one at a time.
  - Draw a flow chart to show the process
  - Code the flow chart in Python language

BNF (Backus-Naur-Form)

- First used to describe the syntax of Algol60.
- BNF is a language for defining the semantics of languages -**metalanguage**
- BNF greatly simplifies semantic specifications.

BNF අංකනය ....

උදාහරණ

<A> := <B> | <C>

<B> := “(“ <D>“)”

<C> := “[“ <D> “]”

<D> :=“a” | “b”| ……..| “z”

BNF අංකනය ....

<identifier> ::= <letter> | <identifier> <letter> |
<identifier> "_"

<letter>::= "a"|..|"z"

BNF අංකනය ....

Python භාෂාවේ ප්‍රකාශවල වාූහය (Syntax) නිර්වචනය කිරීම සඳහා BNF අංකනයේ ව්‍යාප්තියක් (extension) භාවිත වෙයි. මෙම ව්‍යාප්තියේ පහත දැක්වෙන අමතර අනුලක්ෂණ භාවිත කෙරේ.

BNF අංකනය ....

Meta-character	Meaning
*	zero or more repetitions of the preceding item
+	one or more repetitions of the preceding item
[ ]	zero or one occurrences of items inside [ ] - This means what is inside is optional.
( )	Grouping of items
" "	Delimiters for literal strings

BNF අංකනය ....

උදාහරණ

name *::= lc_letter (lc_letter / "_ ")**

lc_letter *::= "a"..."z"*

BNF අංකනය ....

උදාහරණ

name *::= lc_letter (lc_letter | "_ ")**

lc_letter *::= "a"..."z"*

BNF අංකනය ....

උදාහරණ

Program ::= (NEWLINE | statement)*

statement ::= stmt_list NEWLINE | compound_stmt

stmt_list ::= simple_stmt (";" simple_stmt)* [";"]

How to learn a language?

- You must read and write *code*.

What you should know when using a language?

- Structure of a program.
- How comments can be included?
- Structure of the identifiers.
- Variables
- Operators and operator precedence
- Assignments
- Data types
- Control structures
- Input/output
- Sub-programs

Structure of a Python program

- පයිතන් ක්‍රමලේඛයක් එම භාෂාවේ කාරක රීතිවලට අනුකූලව සකස් කළ ප්‍රකාශයන්ගේ එකතුවක් ලෙස දැක්විය හැකි ය. පයිතන් ප්‍රකාශ Simple හා Compound ප්‍රකාශ ලෙස වෙන් කළ හැකි වේ
- **Pythons statements**
  - **Simple statements :** Simple statements are limited to a single logical line.
  - **Compound statements : C**ontain (groups of) statements. Typically, a compound statements span multiple lines.

Structure of a Python program

Program ::= (NEWLINE | statement)*

statement ::= stmt_list NEWLINE | compound_stmt

stmt_list ::= simple_stmt (";" simple_stmt)* [";"]

Nested statements are not allowed in a semicolon-separated list of simple statements on the same line.

Practical Session 03

- Objectives : Identify programs with the correct structure.

Example - 1

#program02.py

Practical Session 03 .....

Example - 2

```
#program03.py
x = 1
y = 2
z = x + y
print(z)
```

Practical Session 03 .....

Example - 3

```
#program04.py
x = 1 y = 2 z = x + y
print(z)
```

Practical Session 03 .....

Example - 4

```
#program05.py
x = 1; y = 2; z = x + y
print(z)
```

Structure of a Python program

උදාහරණ

```
 #Examples.py
#First Example
   while True:
         x=input("Enter a String : ")
         char_count = len(x)       # compute the length
         print ("String; " ,x, " length : ", char_count)
```

Structure of a Python program

- පයිතන් ක්‍රමලේඛයක් තර්කානුකූල පේළි (logical lines) අනුක්‍රමයකින් සමන්විත වෙයි.
  - එක් තාර්කානුකූල ප්‍රකාශයක් (logical line/ statement) එක පයිතන් ප්‍රකාශයක් විය යුතුය.
  - මෙහි එක් තාර්කික උපදෙසක් පේළි කිහිපයක් දක්වා විහිදිය හැකිය.
- මෙවැනි අවස්ථාවන්හීදී Explicit line joining හෝ Implicit line joining නීතිවලට අනුකූලවන ලෙස ප්‍රකාශනයක් පේළි කිහිපයක දැක්විය හැකිය.

Explicit line joining

- ක්‍රමලේඛයක කිසියම් පේළියක් ‘\’ යන අනුලක්ෂයෙන් අවසාන වී ඇත්නම් එම පේළිය ඊට පසුව ඇති පේළිය සමග සම්බන්ධ කරනු ලබයි.

උදාහරණ :

```
x = 1 + \
    2
```

Implicit line joining

(), [] සහ {} යන වරහන් තුළ වූ ප්‍රකාශ, පේළි කිහිපයක වන සේ ඛේත කළ හැකිය.

උදාහරණ :

```
x = {8:'a',9:
'b',10:'c'}
```

- පයිතන් ක්‍රමලේඛ ගොඩනැගීමේ දී ඕනෑ ම වදන් සකසනයක් භාවිත කළ හැකි ය. එනමුදු ප්‍රකාශනයන්ගේ ඉදිරියෙන් ඇති හිස් අවකාශයන්ගේ ඇති වැදගත්කම නිසා Tab යතුර මඟින් පේළි අතර රටාවන් ඇති කළ හැකි වදන් සැකසුම් පයිතන් ක්‍රමලේඛ සම්පාදනය සඳහා වඩාත් සුදුසු වේ.
- පයිතන් ක්‍රමලේඛ සුරැකීම (Save) සඳහා py හෝ pyc යන ගොනු දිගුවන් (Extension) භාවිත කළ යුතු ය.

Executing a Python program

- By using Python IDLE
  - Open the file (File -> Open)
  - Run the program (Run->Run Module or F5)
- At the command line
  - python program.py
- In Unix
  - Adding the statement #!/usr/bin/python as the fist line of the program.
  - Making the program executable
  - Typing the program name at the command line
- In Windows
  - Click the right-mouse button on the file name
  - Select ***Open with*** and select the python interpreter program to open the file.

- එක් පේළියක පයිතන් ප්‍රකාශ එකකට වඩා ඇතුළත් කිරීමට අවශ්‍ය වුවහොත් එම ප්‍රකාශ එකිනෙකට වෙන් කිරීම සඳහා ; සලකුණ භාවිත කළ යුතු ය.

උදාහරණ :

```
X=10
print(10)
```

උදාහරණ

```
X=10 ;print(10)
```

Getting needed information

- help()
- help('print')

Practical Session

- Objective : Familiarity with the help facility provided by the IDE.

  - Explore how the following commands could be used.
    - help()
    - print
    - Input
    - if
    - while
    - keywords

Blank Lines

- හිස් අවකාශ, Tabs හා form feed වලින් පමණක් සමන්විත පේළි ක්‍රමලේඛ ක්‍රියාකරවීමේදී සැළකීමට භාජනය නොවේ.

උදාහරණ

```
x = {8:'a',9:

'b',10:'c'}
```

ප්‍රකාශ විභේදනය (Grouping Statements)

- ක්‍රමලේඛයක ඒකකයක් ලෙස සැලකිය හැකි ප්‍රකාශ අනුක්‍රමයක් කට්ටියක් (block) ලෙස හැඳින්වෙයි. ක්‍රමලේඛයක ඇති මෙවැනි කොටසක් එක ප්‍රකාශණයක් ලෙස සංක්ෂේපනය කළ හැකිය. ක්‍රමලේඛ තුල ඇති ප්‍රකාශ මෙවැනි කට්ටිවලට වෙන්කිරීමට පහසුකම් සළසන ක්‍රමලේඛ භාෂා Block-structured languages ලෙස අර්ථ දැක්වෙයි.

ප්‍රකාශ විභේදනය ....

- විවිධ ක්‍රමලේඛ භාෂාවන් ප්‍රකාශ කට්ටි ලෙස විභේදනය කිරීම සඳහා විවිධ ක්‍රමවේද අනුගමණය කරනු ලැබේ. උදාහරණයක් ලෙස “ALGOL” පවුලට අයත් භාෂා ප්‍රකාශ විභේදනය සඳහා ‘begin’ සහ ‘end’ යන මූලපද යොදා ගන්නා අතර “C” පවුලට අයත් භාෂා මේ සඳහා ‘{‘ සහ ‘}” යන අනුලක්ෂණ යොදා ගනී. Python භාෂාව මෙයට වඩා වෙනස් ක්‍රමවේදයක්, එනම් අනුඡේදනය (Indentation) ප්‍රකාශ විභේදනය සඳහා යොදාගනී.

අනු ඡේදනය (Indentation)

උදාහරණ :

```
def sinhalaexample():
    '''මෙම ශ්‍රිතය මගින් සිංහල භාෂාවේ වචන පයිතන් තුළ
    යොදා ගන්නා  ආකාරය පෙන්නුම් කරයි.
    '''
    ගම් = ['මහරගම','නුවර','ගාල්ල']
    for ගම in ගම්:
        if ගම !='නුවර' :
            print(ගම)
```

අනු ඡේදනය (Indentation)

Python ක්‍රමලේඛ සැකසීමේ දී එක් ඒකකයක් ලෙස සැලකිය යුතු ප්‍රකාශ, සෑම පේළියක් ආරම්භයේ සිටම එකම දුරින් ආරම්භවන ලෙස,එකම රටාවට අනුඡේදනය කළ යුතු වෙයි.

Compound ප්‍රකාශ

- “clauses” එකක් හෝ ඊට වැඩි ගණනකින් සමන්විත වූ පයිතන් ප්‍රකාශනයක් compound ප්‍රකාශයක් ලෙස හැඳින්වෙයි.
- සෑම “clause” එකක්ම ශීර්ෂයක් (header) හා මෙම ශීර්ෂයේ පාලනයට නතු වූ ප්‍රකාශන එකතුවකින් (suite) සමන්විත වෙයි.
  - The suite should consist of at least one statement or the keyword “pass” .
- compound ප්‍රකාශනයන්ගේ සෑම clause එකකම ශීර්ෂය මූලපදයකින් ආරම්භ විය යුතු අතර “:” යන අනුලක්ෂණයෙන් අවසාන විය යුතුය.
- Compound ප්‍රකාශයක සියලුම clause යන්හි ශීර්ෂයන් එකම රටාවකට සිටින ලෙස අනුඡේදනය කළ යුතු වෙයි.

Compound ප්‍රකාශ

උදාහරණ :

```
if x > 100 :
    print('Excellent')
    y = 3
elif x > 50 :
    print('Good')
    y = 2
elif x > 30 :
    print('Must improve')
    y = 1
else :
     print('Fail')
    y = 0
```

විවරණ (Comments)

උදාහරණ

```
#Examples.py
#First Example
   while True:
         x=input("Enter a String : ")
         char_count = len(x)       # compute the length
         Print ("String; " ,x, " length : ", char_count)
```

පයිතන් මූල පද (Key Words/Reserved Words)

- පයිතන් භාෂාවේ ද අනෙක් බොහෝ පරිගණක භාෂාවන් මෙන් මූල පද ලෙස අර්ථ දැක් වූ වචන ලැයිස්තුවක් වේ. ක්‍රමලේඛ ගොඩනැගීමේ දී එම වචනවල භාවිතය ඒවායේ අනුරූප අර්ථයන්ට අනුකූලව සිදු කළ යුතු ය. පයිතන් භාෂාවේ ඇති මූල පද ලැයිස්තුව පහත දැක්වේ.
- Python interpreter uses the keywords to identify the structure of the program/statement.

පරිසීමක Delimiters

- පයිතන් භාෂාවේ අඩංගු විවිධ ටෝකන (token) එකිනෙකට වෙන් කිරීම සඳහා පරිසීමක උපයෝගී කර ගැනේ. පහත දැක්වෙන අනුලක්ෂ්‍ය Python භාෂාවේ පරිසීමක ලෙස අර්ථ දක්වා ඇත.

දත්ත සහ වස්තු (Data and Objects)

- පයිතන් භාෂාවේ දත්ත, වස්තු හැටියට නිරූපණය කෙරේ. සෑම වස්තුවකට ම, අනන්‍යතාවක්, ප්‍රරූපයක් සහ අගයක් ඇත. මෙවන් වස්තුවක් නිර්මාණය පසුව එහි ප්‍රරූපය සහ අනන්‍යතාව වෙනස් නො වේ.
  - When a value is assigned to a variable in, Python an association is created between a variable and an object. This association is called a **reference.**

    Eg : x = 5

State Diagram

හඳුන්වන (Identifiers/names)

පයිතන් ක්‍රමලේඛ භාෂාවේ හඳුන්වන තැනීමේ දී පහත දැක්වෙන නීතිවලට අවනත විය යුතු ය.

- හඳුන්වන භාෂාවක අක්ෂරයකින් හෝ “_” අනුලක්ෂයෙන් ආරම්භ විය යුතුය.
- පළමු අක්ෂරයට පසුව එන අක්ෂර සඳහා ඉහත ආකාරයේ අක්ෂර හෝ 0 සිට 9 දක්වා වූ ඉලක්කම් යොදා ගත හැකි ය.
- නමට තිබිය හැකි අක්ෂර සංඛ්‍යාවේ උපරිම සීමාවක් නොමැත.
- පයිතන් භාෂාවේ විශේෂිත පද හඳුන්වන සේ භාවිත කළ නො හැකි ය.
- ඉංග්‍රිසි භාෂාවේ කුඩා අක්ෂර සහ මහා අක්ෂර එකිනෙකට වෙනස් අක්ෂර ලෙස සළකනු ලැබේ. (Case is Significent) උදාහරණයක් ලෙස (Name) සහ (name) එකිනෙකට වෙනස් විචල්‍යය නාම දෙකක් ලෙස සළකනු ලැබේ.

දත්ත ප්‍රරූප (Data Types)

කලින් නිර්වචනය කළ ලක්ෂණ සහ මෙහෙයවන සහිත අගයයන් සමූහයක් දත්ත ප්‍රරූපයක් ලෙස හැඳින්විය හැකි ය. දත්ත භාවිතයට පෙර ඒවායේ ප්‍රරූප දැන ගත යුතු ය.

දත්ත ප්‍රරූප (Data Types)

```
x = 5
y = 2
z = x + y
print(z)
```

```
x = "a"
y = "b"
z = x + y
print(z)
```

Mutable & immutable data type

පයිතන් භාෂාවේ දත්ත ප්‍රරූප “mutable” හා “immutable” ලෙස වර්ග කල හැක. Mutable ගණයට ගැණෙන දත්ත ප්‍රරූපයන්ට අයත් වස්තූන්ගේ අගයන් එම වස්තු නිර්මාණයෙන් පසු ඕනෑම අවස්ථාවක වෙනස් කළ හැකිය. එනමුදු immutable ගණයට අයත් වස්තූන් නිර්මාණය කල පසු ඒවායේ අගයන් වෙනස් කළ නොහැකිය. Numbers, strings සහ tuple, immutable ගණයට අයත් දත්ත ප්‍රරූප වන අතර lists සහ dictionaries යන ප්‍රරූප mutable ගණයට අයත් ප්‍රරූප වෙයි.

Iterable data types

පයිතන් භාෂාවේ ඇති එකවරට එක අවයවයක් බැගින් මුදා හැරීමට හැකියාවක් ඇති දත්ත එකතුවන් iterable objects ලෙස හැඳින්වෙයි. List, string හා tuple දත්ත ප්‍රරූප මෙවැනි iterable වස්තු ප්‍රරූප සඳහා නිදසුන් කිහිපයකි.

Numbers

මෙමගින් ගණිතමය සංඛයා නිරූපනය වෙයි. තබාගත හැකි කුඩාම හා විශාලම සංඛ්‍යාව පරිගණකයේ සංඛ්‍යා නිරූපණය සඳහා ඇති සීමාවන් මත රඳා පවතී.

Sequences

මෙමඟින් සීමාවක් සහිත අනුක්‍රමයන් නිරූපණය කරයි. මෙම අනුක්‍රමයේ සෑම අගයකටම ඍණ නොවන (non-negative) සංඛ්‍යාවක් මඟින් ප්‍රවේශ විය හැකිය. මෙම අගයන් Index අගයන් ලෙස හැඳින්වෙයි. මෙම වර්ගයේ අනුක්‍රමයක ඇති අගයන් ගණන len( ) යන ශ්‍රිතය ක්‍රියාත්මක කිරීමෙන් ලබා ගත හැකිය. යම් අනුක්‍රමයක ඇති අගයන්ගේ ගණන n නම්, Index කුලකයේ අගයන් 0 සිට n-1 දක්වා වෙයි. *a* [*i*] යන වාූහය මඟින් *a* ලෙස අර්ථ දැක්වූ අනුක්‍රමයේ *i* යනුවෙන් දක්වනු ලැබූ අවයවයේ අගයට ප්‍රවිශ විය හැකිය. මෙම ගණයට ගැනෙන දත්ත ප්‍රරූප immutable හෝ mutable විය හැකිය.

Common operations on sequences

Let s and t be two sequences

- s[i] return the ith element of s, origin is 0
- s[i:j] return the slice of s from i up-to j th element.
- s[i:j:k] return the slice of s from i up-to j th element with step k, the elements would be I,i+k,i+2k,….
- x in s True only when x is an item of s
- x not in s True only when x is not an item of s
- s+t Concatenate two sequences
- len(s) Returns the number of elements in s
- min(s) Returns the smallest item in s
- max(s) Returns the largest item in s
- s.count(x) returns the total number of occurrences of x in s

Indices

- Any integer expression can be used as an index.
- If an attempt is made to read or write an element that does not exist, an error is generated.
- If an index has a negative value, it counts backward from the end of the list.

Strings

- A string is a **sequence of characters enclosed in " and " or ' and ' or ""' and "' if string spans for multiple lines.**
- Strings are immutable**.**
- Any character of a string can be accessed by using an index.
- The index of the first letter in a string is zero(not one).

Example :

x = "school"

print(x[1]) # will print the character "c"

print(x[-1]) # will print the last character "l"

- len(x) # gives the number characters in the string x

Traversing a string

- Traversing – going through all the element.

```
for c in “school”:
        print(c)
```

Strings –Accessing individual characters

Index	0	1	2	3	4	5
String	**S**	**c**	**h**	**o**	**o**	**l**
Index	-6	-5	-4	-3	-2	-1

Checking the existence of a character(item) in a string

- *in* operator returns a Boolean value depending on its existing in the string.

  Example :

  “a” in “Teacher” #Returns True

  “x” in “Teacher” #Returns False

Breaking a string into words (split)

x = “a ab abc def”

list = x.split( ) # split the sentence at space characters to build a list

list = x.split(“,”) # What will happen??

x = “a,ab,abc,def”

list = x.split(‘,’)

String Slicing

- A segment of a string is called a **slice.**
- The operator [n:m] on a string returns the slice of the string from the "n^{th}" character to the "m^{th}" character, including the n^{th} character but excluding m^{th} character.

Example

x ="school"

print(x[1:4]) # print the slice "cho"

print(x[:4]) # ???

print(x[4:]) # ???

print(x[:]) # ???

String methods

- A method is similar to a function(function on an object)
  - It acts on an object arguments and returns a value
- The syntax of an method

  object.method(argument)

Example :

x = “abc”

print(x.upper()) #**invoking** the method *upper* on the object *x*

x.find(‘b') # ????

String Slicing

- If the first index is greater than or equal to the second the result is an **empty string.**

Lists

- A list is a sequence of values.
- Lists are mutable
- The individual values of a list can be of any type.
- The values in a list are called **elements** or **items**.
- List Constriction
  - a = [1,1.2,"abc",[1,2,"c]]
  - a = [] # creating an empty list
- List concatenation (+)

  [1,2,3] + [4,5] # produce the list [1,2,3,4,5]

Lists

- Repeating element of a list (*)

  [1,2,3]* 3 # produces [1,2,3,1,2,3,1,2,3]

- List slicing(:)

  The slicing operator can be used to obtain a part of a list as a new list.

- Checking the existing of an element in a list (in)

  5 in [1,3,5,8] # return True

  2 in [1,3,5,8] # returns False

- Removing elements from a list

  del list[index1:index2]

Lists

- String to a list

  v = list(“abcd”)

Adding elements to a list

- Two different ways to add element/s to a list
  - t.append(x)
  - t = t + [x]

- The following constructs will not add an element to the list. Why ?
  - t.append([x])
  - t = t.append(x)
  - t + [x]
  - t = t + x

Traversing a list (for)

```
for i in [1,2,3]:
    print(i)

x = [1,2,3]
for i in range(len(x)):
    print(x[i])
```

Dictionaries

- A dictionary is like a list, but the indices of a dictionary must be provided by the user an can be of (almost) any type. Whereas in a list indices must be integers.
- The indices of a dictionary are called **keys**.
  - Each key maps to a value.
  - The association of a key and a value is called a **key-value pair** or sometimes **an item.**
- The order of items in a dictionary is unpredictable(not a sequence).

Dictionary operators

- d = dict() # creates an empty dictionary
- d = {}
- d = {1:'abc','t':1} #creates a dictionary with two items
- d['t'] #access the element with the key 't' from the dict. D
- len(d) # number of key-value pairs in the dict.
- 't' in d # Is 't' a key in the dict. D
- t = d.values() # returns the values in the dict. as a list.

Tuples

- A tuple is a sequence of values.
- The values can be any type.
- The values of a tuple are indexed by integers
- The tuples are immutable.

Operators on Tuples

- Tuple construction

  a = 1,'a',[1,2]

  a = (1,'a',[1,2])

- To create a tuple with a single element, a comma after the element must be included

  a = 1,

- A value in parentheses is not a tuple

  a = ('1') # a is not a tuple

- Most list operators also work on tuples

Comparing Sequences

- The relational operators work with sequences.
  - Python starts by comparing the first element from each sequence. If they are equal, it goes on to the next elements, and so on, until it comes across elements that differ. Based on this different values appropriate Boolean value is returned and the subsequent elements are ignored(even if they are really big).

  Example :

  [1,2,4] < [1,2,3,4,5,6,7] # False

  "abcd" <"abdefg“ #True

Set type

මෙමගින් පිළිවෙලක් රහිත සීමාවක් සහිත අගයන්කුලකයක් නිරූපනය වෙයි. මෙම කුලකයේ අගයන්ට ප්‍රවේශ වීම සඳහා index අගයන් භාවිතා කළ නොහැකිය.

- A set is not a sequence

```
a = set()          # creating an empty set
a.add(x)           # adding an element to a set. Why not a +{x} ??
a = {1,2}
b = {2,3}
a.union(b)
a.intersection(b)
a.difference(b)
.............
```

Mappings

- මෙමඟින් සීමාවක් සහිත වස්තු කුලකයක් නිරූපණය වෙයි. මෙම කුලකයේ අගයන්ට ප්‍රවේශ වීම සඳහා කැමති Index දත්ත කුලකයක් යොදාගත හැකිය. *a* [K] යන අංකනය මඟින් *a* ව්‍යුහයේ K යනුවෙන් දැක්වෙන අවයවයට ප්‍රවේශ විය හැකිය

Operations on data types - Examples

- Construction

  a = ['abc', (1,5.6,'cde'), 100, 10.57], a = []

- Finding the number of items
  - Len(a)
- Accessing items

  a[0], a[0:2] # from index 0 up to index 2

  : - slicing operator

  a[-1] # the index -1 will always point to the last value on the list

Operations on data types - Examples

- Modifying items

  a[2] = 'xyz'

- List membership

  'abc' in a

- Deleting elements

  del a[2]

- Adding elements to the end of a list

  a = a + [6,8]

Operations on data types - Examples

- Accessing elements

  a = ['abc', (1, 5.6, 'cde'), 100, 10.57, 6, 'abc', 6, 8]

  a[1][2]

List Slicing

- a[start:end] # items start through end-1
  a[start:] # items start through the rest of the array
  a[:end] # items from the beginning through end-1
  a[:] # a copy of the whole array
  a[-1] # last item in the array
  a[-2:] # last two items in the array
  a[:-2] # everything except the last two items

නියත (Literals or Constants)

- ප්‍රභව කේතවල ඇති නිශ්චිත අගයයන් මේ නමින් හැඳින්වේ. පයිතන් භාෂාවේ ඇති ප්‍රධාන නියත ප්‍රරූප කිහිපයක් නම්
  - ඉපිලෙන ලක්ෂණ (Floating point)
  - නිඛිල (Integer)
  - අනුලක්ෂ දාමය (String)

ඉපිලෙන ලක්ෂ්‍යය නියති (Floating point literals)

floatnumber ::= pointfloat | exponentfloat

pointfloat ::= [intpart] fraction | intpart "."

exponentfloat ::= (intpart | pointfloat) exponent

intpart ::= digit+

fraction ::= "." digit+

exponent ::= ("e" | "E") ["+" | "-"] digit+

උදාහරණ 15.1

3.141, 12. , .42, 14.2e2

විචල්‍යය (Variable)

- ක්‍රමලේඛ ක්‍රියාකරවීමේ දී දත්තයන් තාවකාලික ව ප්‍රධාන මතකයේ තබා ගැනීමට අවශ්‍ය වේ. මෙසේ දත්තයන් මතකයේ තාවකාලික ව තබා ගැනීම සඳහා විචල්‍යය භාවිත වෙයි. පරිගණකයේ ප්‍රධාන මතකයේ නිශ්චිත කොටසකට ප්‍රවේශ වීම සඳහා උපයෝගී කරගත හැකි සංකේත නාමයක් ලෙස විචල්‍යයක් දැක්විය හැකි ය. මෙලෙස විචල්‍යයක් මතකයේ කිසියම් කොටසකට තාවකාලික ව අනුබද්ධ කළ විට එම විචල්‍යය උපයෝගී කර ගෙන එම අදාළ මතක කොටසේ විවිධ දත්ත තාවකාලික ව ගබඩා කිරීම සහ නැවත ලබා ගැනීම සිදු කළ හැකි ය.

විචල්‍ය පිළිබඳ දැනගත යුතු මූලික කරුණු

- පරිගණකයේ ප්‍රධාන මතකයේ දත්ත ගබඩා කිරීම සඳහා වෙන් කර ගත් කොටසකට අනුබද්ධ කළ සංකේත නාමයක් ලෙස විචල්‍යයක් දැක්විය හැකි ය.
- විචල්‍යයක් සඳහා වෙන් කරගත් මතක ඛණ්ඩයට ක්‍රමලේඛය ක්‍රියාත්මක වන කාලය තුළ විවිධ අවස්ථාවන්හි දී විවිධ අගයන් රඳවා තැබිය හැකි ය. මෙලෙස විවිධ අවස්ථාවන්හි දී විවිධ අගයන් ගබඩා කිරීමේ දී අවසාන වශයෙන් ගබඩා කළ අගය මගින් ඊට පෙර ගබඩා කළ අගයක් ඇතොත් එය ප්‍රතිස්ථාපනය වෙයි.
- මෙහි මුලින් i යනුවෙන් හැඳින්වූ මතක ඛණ්ඩයේ ගබඩා කළ 10 යන අගය දෙවැනි ප්‍රකාශය ක්‍රියාත්මක වීමේ දී 5 යන අගයෙන් ප්‍රතිස්ථාපනය වෙයි. අවසානයේ දී **i** යනුවෙන් හැඳින්වෙන මතක ඛණ්ඩයේ ගබඩා වී ඇති අගය 5 වේ.
- මතකයේ කොටස් විචල්‍යය සඳහා අනුබද්ධ කිරීම තාවකාලික ක්‍රියාවලියකි. ක්‍රමලේඛය ක්‍රියාත්මක කර අවසන් වූ විට එම ක්‍රමලේඛය මගින් විචල්‍යය සඳහා ලබාගත් මතකයේ සියලු කොටස් නැවත පරිගණකයට මුදා හරි.
- විදුලිය විසන්ධි කිරීමක දී විචල්‍යය ප්‍රධාන මතකයෙන් ඉවත් ව යයි.

පයිතන් භාෂාවේ වලංගු විචල්‍යය නාමයන්ට උදාහරණ කිහිපයක් පහත දැක්වේ.

- _name
- A5
- My _Name
- _Name5
- ගම
- නම

- පයිතන් භාෂාවේ වලංගු නොවන විචල්‍යය නාමයන්ට උදාහරණ කිහිපයක් පහත දැක්වේ.

  - 98
  - 2 Name
  - -Name
  - Name e
  - Name@

මෙහෙයවන (Operators)

මෙහෙයවන, දත්තයන් මත කළයුතු ක්‍රියාවන් නිරූපණය කරයි. Python භාෂාවේ මෙහෙයවන ප්‍රධාන ඛාංඩ කිහිපයකට බෙදිය හැකිය

අංක ගණිත පරිවර්තන (Arithmetic conversions)

එකිනෙකට වෙනස් ආකාරයේ සංඛ්‍යා දෙකක් මත ද්විමය ගණිත කර්ම යොදා ගැනීමේදී පහත සඳහන් නීති භාවිත කෙරේ.

- එම සංඛ්‍යා දෙකෙන් එකක් සංකීර්ණ සංඛ්‍යා නම් අනෙක් සංඛ්‍යාව ද සංකීර්ණ සංඛ්‍යාවකට හරවනු ලැබේ.
- එසේ නැතිනම් එම සංඛ්‍යා දෙකෙන් එකක් ඉපිලෙන සංඛ්‍යා නම් අනෙක් සංඛ්‍යාව ද ඉපිලෙන සංඛ්‍යාවකට හරවනු ලැබේ.
- එසේත් නැති නම් එම සංඛ්‍යා පූර්ණ සංඛ්‍යා විය යුතු යි. එවිට සංඛ්‍යා පරිවර්තනය වීමක් නොවේ.

ප්‍රමුඛතා නීති (Precedence Rules)

Example :

2- 3 + 5*5 + 4 ** 2

සංඝටකාව (Associativity)

Example :

2 – 3 + 5

ඇගයීමේ පිළිවෙත වෙනස් කිරීම (Changing the order of evaluation)

- ඇගයීමක මූලික පිළිවෙළ වරහන් යෙදීම මගින් වෙනස් කළ හැකි ය. එම වරහන් යම් අයිතම කාණ්ඩයක් සඳහා යොදා ගන්නේ නම් එම වරහන් තුළ ඇති පදය වඩාත් අභ්‍යන්තරයේ ඇති වරහන්වලින් පටන් ගෙන පිටත වරහන්වල ඇති පදය දක්වා ඇගයීම සිදු කෙරේ

  Example :

  ((2 + (3 – 5)) **2)//4

Example

Consider the following list

[[‘a1’,28],[‘a2’,43],[‘a3’,0],[‘a4’,12]]

Each inner list in the above list keep the name and marks obtained by a student at a particular examination.

*Write a Python program to compute the average mark of the marks obtained by all students.

Solution

```
x=[['a1',28],['a2',43],['a3',0],['a4',12]]
count = 0;
tot = 0;
for std in x:
    name,mark = std
    count = count + 1
    tot = tot + mark;
print("Student Count =",count);
print("Average mark = ",tot/count);
```

පැවරුම් ප්‍රකාශන (Assignment statements)

විචල්‍ය සඳහා අගයන් බද්ධ කිරීමට හෝ වෙනස් කිරීමට හෝ වස්තුවක ගුණාංග වෙනස් කිරීමට පැවරුම් ප්‍රකාශය යොදා ගනී.

- කාරක රීති :

  assignment_stmt ::= (target_list "=") $^{+}$ (expression_list)

  target_list ::= target ("," target)* [","]

- **Semantic:**

  ප්‍රකාශ ලැයිස්තුවක ඇති ප්‍රකාශ කොමාවන්ගෙන් වෙන් කරන ලැයිස්තුවක් නම් එම ප්‍රකාශ ඇගයීම මගින් tuple යක් සැකසේ. පැවරුම් ප්‍රකාශනයක් මගින් expression list හි ඇති ප්‍රකාශ ඇගයීමට භාජනය කර ලැබෙන ප්‍රතිඵලය ලැයිස්තුවේ එක් එක් අයිතම සඳහා වමේ සිට දකුණට ආදේශ කරයි.

පැවරුම් ප්‍රකාශන (Assignment statements)

Examples :

```
mylist = [1,2,3]
a,b = 2*3,(3,4)  # result is a tuple
a,b = c,d = True and False,2
a,b = "12"
add =ddk@ucsc.ac.lk
name,domain = add.split('@')
```

- The right side can be any kind of sequence (string, list or tuple).

  Note : In Python prior to updating a variable, it must be **initialized.**

ගැලීම් පාලන ව්‍යුහයන් (Flow of Execution)

ක්‍රමලේඛයක් ක්‍රියාකරවීම ක්‍රමලේඛයේ පළමු වගන්තියෙන් ආරම්භ වෙයි. ඉන් පසු ක්‍රමලේඛයේ වගන්ති දක්වා ඇති අනුපිළිවෙළට, වරකට එක් වගන්තිය බැගින්, ඉහළ සිට පහළට ක්‍රියාත්මක කෙරේ. මෙසේ සිදුවන ස්වයන් පැවරුම් if, while සහ for යන ගැලීම් පාලන ව්‍යුහයන් මගින් වෙනස් කළ හැකි ය.

if ප්‍රකාශ

if ප්‍රකාශය කොන්දේසිගත අනුකලනය සඳහා භාවිත කෙරේ.

```
if_stmt ::=  "if" expression ":" suite
        ( "elif" expression ":" suite )*
        ["else" ":" suite]
```

if ප්‍රකාශ

```
if x > 100 :
    ptint('Excellent')
    y = 3
elif x > 50 :
    print('Good')
    y = 2
elif x > 30 :
    print('Must improve')
    y = 1
else :
     print('Fail')
    y = 0
```

Iterative control structures

- Controlled by a logical expression
- Controlled by a counter
- Controlled by an iterator

While ප්‍රකාශ

යම් උපදෙස් මාලාවක් කිසියම් ප්‍රකාශනයක් සත්‍ය වන තෙක් නැවත නැවත කිරීම සඳහා මෙම ව්‍යුහය භාවිත කෙරේ.

කාරක නීති මාලාව

while_stmt ::= "while" expression ":" suite
["else" ":" suite]

While ප්‍රකාශය

```
i = 0
x = []
while i < 5:
    a = int(input("Enter a String : "))
    x = x + [a]
    i = i + 1
print(x)
```

For ප්‍රකාශ

මෙම ප්‍රකාශනය පුණරාවර්තනය කළ හැකි අනුක්‍රමයක සෑම අවයවයක් මතම කිසියම් උපදෙස් මාලාවක් ක්‍රියාකරවීම සඳහා යොදා ගැනේ.

for_stmt ::= "for" target_list "in" expression_list ":" suite
["else" ":" suite]

For ප්‍රකාශ

```
ගම් = ['මහරගම','නුවර','ගාල්ල']
for ගම in ගම්:
    print(ගම)
```

range built-in function

range([*start*], *stop*[, *step*])

- creates iterables yielding arithmetic progressions.
- arguments must be integers.
- If the *step* argument is omitted, it defaults to 1.
- If the *start* argument is omitted, it defaults to 0.
- The full form returns an iterable of integers [start, start + step, start + 2 * step, …].

range(8)

range(2,8)

range(2,8,2)

range (8,3,-1)

range built-in function

```
for i in range (3,8):
    print(i)

for i in range (8,3,-1):
    print(i)
```

Traverse a sequence items with their indices(enumerate)

```
for index, element in enumerate('abc'):
        print(index, element)
```

ශ්‍රිත (Functions)

- **A function** can be viewed as a named sequence of statements that performs a computation.

ශ්‍රිත (Functions)

- ශ්‍රිතයක් යනු අවශ්‍ය කාර්යයක් හෝ කාර්යයන් කර ගැනීම සඳහා සම්පාදනය කල ප්‍රකාශ අනුක්‍රමයකි (Sequence).
- ශ්‍රිතයක අදාළ කාර්ය හෝ කාර්යන් ශ්‍රිතයේ නිර්වචනය (Definition)මඟින් දැක්වෙයි.
- ශ්‍රිත මගින් ක්‍රමලේඛයක ඇති කොටසක් තනි ඒකකයක් ලෙස උපුටා ගෙන එමකොටස, එම ක්‍රම ලේඛයේම හෝ වෙනත් ක්‍රමලේඛයක නැවත නැවත භාවිතයට ගත හැකි පසුබිමක් සකස් කර දෙයි.
- ක්‍රමලේඛයක එකම ඛේත නැවත නැවත ලිවීමේ අවශ්‍යතාවය ශ්‍රිත මගින් දුරු කරයි.
- පයිතන් භාෂාවේ කලින් සකස් කරන ලද මෙවැනි ශ්‍රිත විශාල සංඛ්‍යාවක් ඇත. එමෙන් ම තමන්ට අවශ්‍ය ශ්‍රිතයන් ගොඩනඟා ගැනීමේ අවස්ථාව ද සලසා ඇත.

ශ්‍රිත (Functions) ...

පයිතන් භාෂාවේ ශ්‍රිතයක ව්‍යුහය පහත පරිදි වේ.

```
def function_name(parameter list):
    suite
```

ශ්‍රිතයේ නිර්වචනය (Definition)

- මෙහි **def** යන මූලපදය **මගින් ශ්‍රිතයේ** නිර්වචනය (definition) ආරම්භ කරයි.
- ශ්‍රිතයේ නම def මූලපදයට පසුව ලිවිය යුතු අතර ඉන් පසුව වරහන් තුළ ඇති විධිමත් පරාමිති (parameters) ලැයිස්තුව අවසන් කළ යුත්තේ ":" සංකේතයෙනි.
- මෙම පේළිය ශ්‍රිතයේ "Header" ලෙස හඳුන්වනු ලැබේ. ":"සංකේතයෙන් පසුව එන කොටස ශ්‍රිතයේ Body ලෙස හඳුන්වනු ලැබේ. මෙම Body හි වගන්ති අනුච්ඡේදනය (Indent) විය යුතුයි.

- ශ්‍රිතයේ නම හඳුන්වනයකි (Identifier). එබැවින් ශ්‍රිත නාම යෙදීමේදී හඳුන්වන සඳහා වන නීති භාවිත කළ යුතු ය.
- විධිමත් පරාමිති (formal Parameters), ශ්‍රිතයට අගයන් යැවීම සඳහා අවශ්‍ය යාන්ත්‍රණය සපයන අතර මෙමගින් ශ්‍රිතයේ ක්‍රියාකාරිත්වය පාලනය කිරීම සිදු කළ හැකි ය.
- මෙම පරාමිති ලැයිස්තුව (Parameter List) ඕනෑම පරාමිති සංඛ්‍යාවකින් යුක්ත ද විය හැකි අතර අවයව රහිත හිස් ලැයිස්තුවක් ද විය හැකි ය.

- ශ්‍රිතයක් ක්‍රියාත්මක වන්නේ එම ශ්‍රිතය කැඳ වූ විට ය.
- ශ්‍රිතයක් කැඳවීමේ දී **ශ්‍රිතයේ අඩංගු විධාන ක්‍රියාත්මක වන අතර** සෑම විටම අගයක් මුදා හරී.
- ශ්‍රිතයකට වෙනත් ශ්‍රිතයන් ද කැඳවිය හැකි ය.
- ශ්‍රිතයක් ක්‍රියාත්මක කිරීමට පෙර එය නිර්මාණය (create) කළ යුතුය. එනම් ශ්‍රිතයක් කැඳවීමට පෙර ශ්‍රිත නිර්වචනය(Function Definition) ක්‍රියාත්මක කළ යුතු ය.
- A function can return a tuple.

Functions returning multiple values

Example:

```
def max_min(a,b):
        if a > b:
                return (a,b)
        else:
                return (b,a)
```

ශ්‍රිත කැඳවීම (Function Call)

- ශ්‍රිතයක් ක්‍රියාත්මක කිරීමට එම ශ්‍රිතය කැඳවිය යුතු ය.
- ශ්‍රිත කැඳවුමක දී ක්‍රියාවට නංවන ශ්‍රිතයේ නම හා අදාළ විස්තාර (arguments)ලැයිස්තුව ලබා දිය යුතු ය.

ස්වයං පැවරූ විස්තාර අගයයන් (Default Argument Values)

- ශ්‍රිත නිර්වචනයේ දී විධිමත් පරාමිති සඳහා ස්වයං පැවරූ අගයයන් පැවරිය හැකි ය.
- මෙමඟින් ශ්‍රිත නිර්වචනයේ සඳහන් වූවාට වඩා අඩු විස්තාර ගණනක් සහිතව ශ්‍රිත කැඳවීමට ද ඉඩ ලබා දේ.
- විධිමත් පරාමිතියකට අගයයන් නොදෙන අවස්ථාවල දී එම පරාමිතිය සඳහා ස්වයං පැවරූ අගයක් උපකල්පනය කරනු ලබයි. එම ස්වයං පැවරූ අගය එක්වරක් පමණක් ඇගයීමට ලක් වන අතර ඉන් අනතුරුව කරන කැඳවීම්වලදී ද එම අගයන් යොදා ගනී.

Example

```
def add(a,b = 100):
    print("First parameter",a)
    print("Second parameter",b)

print("--Position mapping-------")
add(12,5)
print("--Key word mapping-------")
add(a=12,b=5)
print("--Key word mapping-------")
add(b=5,a=12)
print("--- Impact of default values----")
add(6)
```

All arguments without default values must be listed before arguments with default values in the function definition.

Any argument can be passed either implicitly by position or explicitly by name.

විස්තාර සමඟ විධිමත් පරාමිති බන්ධනය (Binding arguments with formal parameters)

- ස්ථානය හෝ මූලපදය අනුව විධිමත් පරාමිති සඳහා අගයයන් පවරන ක්‍රම දෙකක් ඇත.
- විධිමත් පරාමිති සඳහා ස්ථානය අනුව අගයයන් පැවරීමේ දී ශ්‍රිත කැඳවුම් විස්තාරයන් සහ ශ්‍රිත නිර්වචනයේ විධිමත් පරාමිති අතර එකට එක අනුරූපණයක් තිබිය හැකි ය.
- මූලපද භාවිත කොට අගයන් බන්ධනය කිරීමේ දී ශ්‍රිත කැඳවුමේ ඇති විස්තාරයන් Keyword=value ආකාරයට විය යුතු ය. මෙසේ යොදා ගන්නා මූලපද (key word) විධිමත් පරාමිතිවල නාමයන් විය යුතු ය.
- ශ්‍රිත කැඳවීමක දී විස්තාර ලැයිස්තුවේ අවයවයන් පයිතන් භාෂාවේ පරාමිති සමඟ සමුද්දේශ යොමුවන් මගින් (Reference) බන්ධනය වෙයි. එනම් ශ්‍රිතයක් තුළ ඇති පරාමිතික අගයන් වෙනස් කළ විට එම වෙනස, කැඳ වූ ශ්‍රිතයේ ද එලෙස ම දැක්වෙයි

- ශ්‍රිතයක් කැඳවන සෑම අවස්ථාවක ම අගයක් මුදා හරී.
- ශ්‍රිතයෙන් මුදා හරින අගය ගම්‍යව නිර්වචනය (explicitly specified) නොවේ නම් “None” යන අගය ලබා දේ. එලෙසම මුදාහරින වගන්තියක් ශ්‍රිතය තුළ නොමැති විට “None” යන්න මුදා හරී. return යන පයිතන් ප්‍රකාශනය කැඳවුම් ශ්‍රිතයට විශේෂිත අගයක් (object) මුදා හැරීම සඳහා භාවිත කළ හැකි ය.

Functions accepting variable number of parameters

- A parameter name that begins with * gathers all unmatched positional arguments into a tuple.

Example :

```
def f1(a,*b):
        print("a : ",a)
        for x in b:
                print("b - ",x)
```

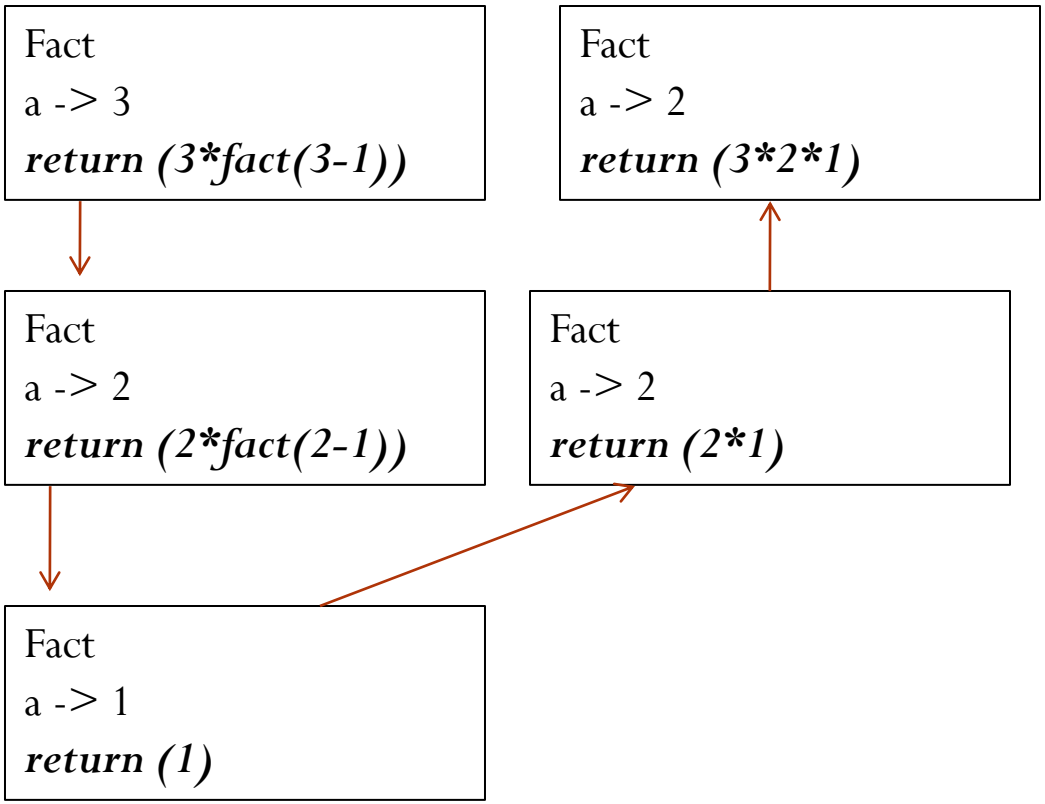
ආවර්තික ශ්‍රිත (Recursive Functions)

- ශ්‍රිතයක් තුළ එම ශ්‍රිතයම නැවත කැඳවීමක් ඇති නම් එම ශ්‍රිතය ආවර්තික ශ්‍රිතයක් (Recursive Functions) ලෙස අර්ථ දැක්වේ.
- ආවර්තිත ශ්‍රිතයක ක්‍රියාකාරිත්වය නැවැත්වීම සඳහා එහි අවසන් කිරීමේ කොන්දේසියක් තිබිය යුතු ය. එසේ නොවුවහොත් ශ්‍රිතය නො නැවතී නැවත නැවත ක්‍රියා කරවීම සිදු වේ. එසේ වුවහොත් ක්‍රමලේඛය බිඳ වැටීමට හෝ පරිගණකය නතර වීමට ඉඩ තිබේ.

ආවර්තික ශ්‍රිත (Recursive Functions)

```
def fact(a):
    if a ==1:
        return 1
    else:
        return (a*fact(a-1))
```

Call graph for recursive functions



ස්ථානීය සහ ගෝලීය විචල්‍යයන් (local and Global Variables)

- ශ්‍රිතයක් තුල ඇති විචල්‍යයන් ගෝලීය විචල්‍යය හෝ ස්ථානීය විචල්‍යයන් ලෙස වර්ග කළ හැකි ය.
  - ගෝලීය විචල්‍යයන් ශ්‍රිතය තුළ දී සහ ඉන් පිටත දී භාවිත කළ හැකි ය. නමුත් ස්ථානීය විචල්‍යයන් භාවිත කළ හැක්කේ ශ්‍රිතය තුළ දී පමණි.
- පයිතන් භාෂාවේ විචල්‍ය සඳහා අගයන් මුලින් ම ලබා දෙන (Initialize) ස්ථානය අනුව එම විචල්‍යයේ සීමාව (Scope) තීරණය කරනු ලැබේ.
  - ශ්‍රිතය තුළ විචල්‍ය ආරම්භ කළේ නම් එම විචල්‍යය ස්ථානීය විචල්‍යයක් ලෙස ක්‍රියා කරයි. එසේ නොමැති නම් එම විචල්‍ය ගෝලීය විචල්‍යයක් ලෙස සලකනු ලබයි.

ස්ථානීය සහ ගෝලීය විචලායයන් (local and Global Variables)

```
#program :varscope.py
i =5
def varscope():
   i =8
   print (i)

varscope()
print (i)
```

ස්ථානීය සහ ගෝලීය විචලායයන් (local and Global Variables)...

- In Python, if a global value is mutable, it can be modified without declaring it

Example :

```
d = {1:'abc', 2:'lmn'}

def example4():
        d[2] = 1
```

- However if a mutable variable has to be reassigned a new value, it must be declared as a global variable.

Reading and Writing data from files

Typical file operations

- Open a file
- Read/Write data
- Close file

Writing data to a file

```
f = open("d:/PythonNIE/workshop/data.txt","a")
data = input("Enter data :")
while(data != ""):
    f.write(data+"\n")
    data = input("Enter data :")

f.close()
```

Reading data from a file(Method 1)

```
f = open("d:/PythonNIE/workshop/data.txt","r")
line = f.readline().strip()
while(line):
    print(line)
    line = f.readline().strip()
f.close()
```

Reading data from a file(Method 2)

```
f = open("d:/PythonNIE/workshop/data.txt","r")
lines = f.readlines()
for line in lines:
    line = line.strip()
    print(line)
f.close()
```

මොඩියුල (Modules)

- A python application typically comprises of code organized in several files – Modules
- A module is a file containing Python definitions and executable statements.
- The executable statements are intended to initialize the module. They are executed only the *first* time the module is imported.
- Modules can import other modules.
- Generally all modules required for a script or module are imported at the beginning , but this is not a requirement of the language.
- The script file name of a module is *module_name*.py

මොඩියුල (Modules) .....

- Creating a Module
  - Create a python program (script) and store the script with the extension .py
- Using a Module
  - Import the module
  - Use the visible items in the module by using the relevant naming conversion.

Import statement

Many varients

- from fmodule import fun1, fun2,.....
- from fmodule import *
  - Imports all names except those beginning with an underscore (_).

Functions in the module can be called as fun1(.....)

In general the practice of importing * from a module or package is not advised as it imports all identifiers in the module to the local name space, possibly hiding already defined items in the current file.

Executing modules as scripts

python module.py <arguments>

The Module Search Path

- When a module named *module* is imported, the interpreter searches for a file named *module.py* in the following directories
  - containing the input script and then
  - in the list of directories specified by the environment variable PYTHONPATH.
  - The search path can be modified by changing the variable sys.path, which is of type list. This variable is defined in the module *sys*

The Module Search Path .....

Example

```
import sys
sys.path = sys.path + ['E:\\AL\\bookexamples\\sdir']

from pathexample1 import *
example1()
```

Reloading a Module

- For efficiency reasons, each module is imported once per interpreter session.
- If you modify a module it can be re-imported by reloading the script.

```
import imp
imp.reload(max)
```

Compileall Module

If a file called *module.pyc* exists in the directory where *module.py* is found, this is assumed to contain an already-"byte-compiled" version of the module *module*.

A module in a directory can be compiled by using the compileall module.

python -m compileall module.py

Boolean expressions

- Expressions that simplify to the Boolean values True or False,
- True and False are not strings, but are the only two values that belong to the type 'bool'.
- In Python any nonzero number is interpreted as "true."

Relational Operators

- A relational operator returns a Boolean value.
  - y == y # x is equal to y
  - x != y # x is not equal to y
  - x > y # x is greater than y
  - x < y # x is less than y
  - x >= y # x is greater than or equal to y
  - x <= y # x is less than or equal to y

  - =< or => are not valid relational operators

Logical operators

- Boolean expressions can be combined together by using the logical operators. In Python there are only three logical operators.
  - and
  - or
  - not

Conditional Execution

- Conditional execution allows a collection of statements to be executed only when a given condition(s) is/are satisfied.

```
if a > 0 :
        print("Greater than 0")
        a = a -1
```

Alternative Execution

- Allows to organize different collections of statements within a single structure where only one collection is executed when a given condition(s) is/are satisfied.

```
if a%2 == 0:
print("The number is even")
else:
        print("The number is odd")
```

- The alternatives are called **branches,** because they provide alternative paths for execution.

Alternative Execution

- Allows to organize different collections of statements within a single structure where only one collection is executed when a given condition(s) is/are satisfied.

```
if x < y:
        print 'x is less than y'
elif x > y:
        print 'x is greater than y'
else:
        print 'x and y are equal'
```

- If more than one condition is true, only the first true branch is executed.

Files

- Text File - A sequence of characters stored on a permanent medium.
- Typical operations
  - Open a file
  - Read/write data
  - Close a file
- Every running program has a "current directory," which is the default directory for most operations.
  - A **relative path** starts from the current directory
  - An **absolute path** starts from the topmost directory in the file system.

Format Strings

- Format a collection of data values.
- The first operand is the **format string,** which contains one or more format sequences, which specify how the second operand is formatted.
- The result is a string.

Example :

```
print('The result of {}* {}is {}' .format (2,.4,2*.4))
print('The result of {0}* {1}is {2}' .format (2,.4,2*.4))
print('The result of {0:3}* {1:4}is {2:5}' .format (2,.4,2*.4))
```

Number of spaces to take
String aligned to left and numbers align to right

Format Strings ....

```
print('The result of {0:4.2f}* {1:4}is {2:5}' .format (2,.4,2*.4))
```

2 decimal places

```
print('The result of {:4.2f}* {:4}is {:5}' .format (2,.4,2*.4))
```

Exception handling

```
try:
   fin = open(‘abc.txt')
   for line in fin:
         print line
   fin.close()
except:
   print 'Something went wrong.'
```

User defined types

- A user-defined type is also called a **class.**
- Class definition in Python

```
class ClassName(Parameters):
    """“Doc String"""“
```

Creating an instance of a class(Object)

- x = ClassName(parameters)
- The return value is a reference to a ClassName object.
- Creating a new object is called **instantiation,** and the object is an **instance** of the class.
- A named values associated with an object is called it's **attribute.**